LOGICAL OPERATIONS ON BITS

So far

* Bits interpreted as integers
* Arithmetic operations on bits

Today

* Bits interpreted as logical values
* Logical operations on bits
Logical values

True $\rightarrow 1$ (convention)
False $\rightarrow 0$

Logical operations

Also known as Boolean functions

A Boolean function $F$ with operands $A$ and $B$ can be specified by a truth table:

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$F$</th>
</tr>
</thead>
</table>

(Binary function)

We can also have unary operations:

<table>
<thead>
<tr>
<th>$A$</th>
<th>$F$</th>
</tr>
</thead>
</table>
NOT function
**AND function**

Truth table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>AND</th>
</tr>
</thead>
</table>

Symbol:

When applied to bit patterns, do it bit by bit:

```
  1 1 0 1 1 0 1 0  AND
  1 1 1 1 0 0 0 0 0
```

AND allows masking out bits to isolate a particular location while ignoring other bits. To do that, we use a bitmask.

**OR function (inclusive OR)**

Truth table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>OR</th>
</tr>
</thead>
</table>

Symbol:

When applied to bit patterns, do it bit by bit:

```
  1 1 0 1 1 0 1 0  OR
  1 1 1 1 1 0 1 0 0
```
Logical operations on bits

1101 1010 OR
1111 0000

XOR function (exclusive OR)

Truth table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Symbol:

When applied to bit patterns, do it bit by bit:

0101 XOR 0101 XOR 0111

Special use: two bit patterns are identical if XOR output is all zeros
Summary

<table>
<thead>
<tr>
<th>Function</th>
<th>Notation</th>
<th>Explanation</th>
<th>Schematic</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>$A \land B$</td>
<td>the “all” function: result is 1 iff all input operands are equal to 1</td>
<td>flat input, round output</td>
</tr>
<tr>
<td></td>
<td>$A \bullet B$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$A \times B$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>$A \lor B$</td>
<td>the “any” function: result is 1 iff any input operand is equal to 1</td>
<td>round input, pointed output</td>
</tr>
<tr>
<td></td>
<td>$A + B$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$A \lor B$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>$\neg A$</td>
<td>logical complement/negation: NOT 0 is 1, and NOT 1 is 0</td>
<td>triangle and circle</td>
</tr>
<tr>
<td></td>
<td>$\bar{A}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR exclusive OR</td>
<td>$A \oplus B$</td>
<td>the “odd” function: result is 1 iff an odd number of input operands are equal to 1</td>
<td>OR with two lines on input side</td>
</tr>
</tbody>
</table>

We will only use the notation highlighted in RED
Logical completeness

How many Boolean functions exist for N variables?

* For one variable (x):

* For two variables (x, y):
* For $N$ variables:

If there are so many functions, why NOT, AND, OR are so important?

Because we can implement any Boolean function with only those 3 functions! (Logical completeness)
Question: can you memorize the following 15-bit pattern in ten seconds?

0 1 1 0 0 0 1 0 1 0 1 1 1 1
Hexadecimal notation

1) Group bits in sets of 4 bits each

2) With 4 bits we can represent

\[ 2^4 = 16 \text{ numbers} \]

Use digits 0-9 and letters A-F
to represent a set of 4 bits

Example: for the bit pattern above

1) 0111 1000 1010 1111

2) \times \quad \quad \quad \quad \quad \quad \quad